Graph Based Pattern Recognition

Exercise 3

Basis

- Chapters 5 and 6

Submission

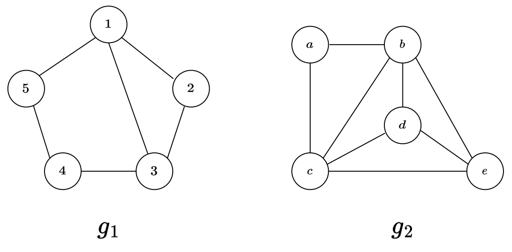
- The submission takes place online on ILIAS.
- Solutions to the theory tasks must be submitted as `*.pdf` file. Other formats will not be accepted.
- Source code for the implementation tasks must be submitted as `*.py` files. Source code that cannot be executed will not be accepted.
- Individual submissions or submissions in teams of two are allowed (hand in only one copy per group). In the source code file, include the *names and matriculation numbers* of both group members in the first two lines as comments.

Dates

- Briefing: 29.03.23
- Submission: 19.04.23
- Debriefing: 19.04.23

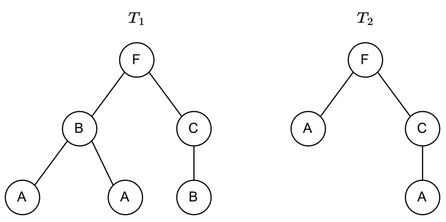
Theoretical Tasks

1. Compute eigendecompositions of the two nearly isomorphic graphs g_1 and g_2, compute $\bar{U}_{g_1}\bar{U}_{g_2}^T$ and find assignment with LSAP solver (use numpy and scipy). Write down all intermediate matrices that result from the whole process.



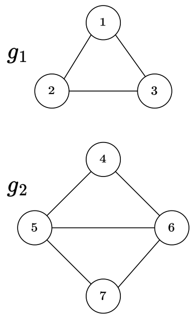
2. Enumerate the basic steps of (a) spectral graph matching and (b) continuous graph matching in a table with two columns. Elaborate on the fundamental differences and similarities of the two approaches for graph matching

3. Compute the tree edit distance between the two trees T_1 and T_2. The cost of insertion or deletion of a node n is defined as $c(n \to \epsilon) = c(\epsilon \to n) = 1$. The cost of substituting one node to another is 0 if both labels are identical and 1 otherwise.



4. Given are two graphs g_1 and g_2 with three and four nodes, respectively. Assume that all nodes of both graphs are embedded in a vector space (e.g., by means of spectral embedding as discussed in Section 5.1.2) and we already have computed the pairwise dissimilarities between all nodes (see Table D below).

   Perform an agglomerative hierarchical clustering based on the dissimilarity matrix D using single linkage (refer to the Appendix for details). To this end, sketch the corresponding dendrogram and then analyze the clustering result and identify the clustering so that at most three nodes are present per cluster. Formalize the resulting many-to-many node mapping defined by this particular clustering.



D

	1	2	3	4	5	6	7
1	0	0,9	1,43	0,81	0,87	1,32	0,1
2		0	1,15	0,23	0,15	1,1	0,81
3			0	1,45	1,21	0,2	1,67
4				0	0,25	1,2	0,85
5					0	1,13	0,8
6						0	1,23
7							0

Implementation Tasks

This practical exercise series consists of two tasks: computing String Edit Distance (SED) and using a genetic optimization algorithm to find the permutation matrix that minimizes a given cost. First, go to the ILIAS's webpage of the course and download/unzip `Exercise_3.zip` in your `PR_Lecture` folder.

1. In the first task, your goal is to implement the String Edit Distance (SED) (Alg. 6 in lecture notes). Navigate to `PR_lecture/Exercise_3/ex3_a.py` and complete the missing part of the source code.

   The alphabet Σ is restricted to lowercase and uppercase Roman letters, represented by the set $\{a, b, c, \dots, z, A, B, C, \dots, Z\}$. For any letter l in Σ, the cost of insertion or deletion of l is defined as $c(l \rightarrow \epsilon) = c(\epsilon \rightarrow l') = 1$. The cost of substituting one letter l for another letter l' is 0 if both letters are identical and the same case (e.g., 'A' == 'A', 'v' == 'v'), 1 if the letters are the same but have different capitalization (e.g., 'a' == 'A', 'V' == 'v'), and 2 if the letters are completely different.

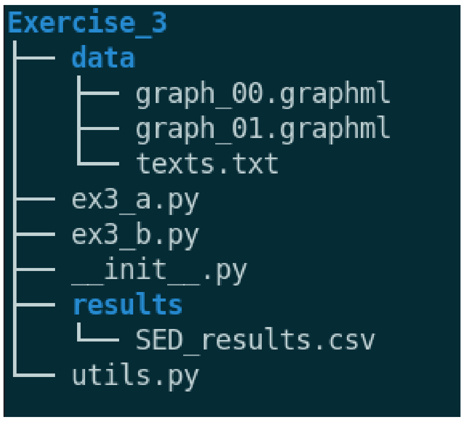
   Load the words to compare from `PR_lecture/Exercise_3/data/texts.txt`. Compute the SED between the pair of words/texts present on each line. Save your results in `PR_lecture/Exercise_3/results/SED_results.csv`

2. In the second task, your goal is to code your own genetic algorithm to find a permutation matrix P that minimizes the following cost between the adjacency matrix A_1 of graph g_1 and the adjacency matrix A_2 of g_2 $||A_1^T P - P A_2^T||_2^2$.

   Based on Alg. 5 from the lecture notes, your implementation generate an initial population of N permutation matrices P and selects the top-performing permutation matrices based on their individual fitness score. The choice of crossover technique is up to you, with options including partially mapped crossover (PMX), cycle crossover (CX), and order crossover operator (OX). You may also choose from several mutation techniques, such as exchange mutation operator (EM), insertion mutation operator (ISM), displacement mutation operator (DM), and simple-inversion mutation operator (SIM).

   After completing the development of your genetic algorithm, load the two graphs located in `PR_lecture/Exercise_3/data`. Experiment with various parameters of your algorithm, such as the number of chromosomes (N), mutation rate, crossover rate, and number of iterations, to find the permutation matrix P' that satisfies the condition $||A_1^T P' - P' A_2^T||_2^2 \leq 4.0$.

Submission
For the coding part, you must submit a `.zip` file containing the following files. Additionally, include your solution for the theoretical tasks as `*.pdf` in the same `.zip` file.



```
Exercise_3
├── data
│   ├── graph_00.graphml
│   ├── graph_01.graphml
│   └── texts.txt
├── ex3_a.py
├── ex3_b.py
├── __init__.py
├── results
│   └── SED_results.csv
└── utils.py
```

Appendix

In this appendix, we consider one concrete algorithm for clustering feature vectors – the *agglomerative hierarchical clustering*. The basic idea of this clustering approach is to start with the data objects as individual clusters and then merge the nearest pair of clusters at each step (this requires the definition of a cluster proximity – see below). The basic form of agglomerative hierarchical clustering is formally represented in Algorithm 1.

Algorithm 1 Agglomerative Hierarchical clustering($X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$)

```
1: $R_0 = \{C_1 = \{\mathbf{x}_1\}, \dots, C_N = \{\mathbf{x}_N\}\}$
2: $t = 0$
3: repeat
4:     $t = t + 1$
5:     take all clusters from $R_{t-1}$ to $R_t$
6:     determine the one with the smallest cluster distance among all possible pairs of clusters
       $C_r, C_s$ in $R_t$: $d(C_i, C_j) = \min\{d(C_r, C_s) | C_r \neq C_s; C_r, C_s \in R_t\}$;
7:     define a new cluster $C_q = C_i \cup C_j$
8:     remove $C_i$ and $C_j$ from $R_t$
9:     insert $C_q$ into $R_t$

10: until all $\mathbf{x}_i \in X$ belong to the same cluster $C$
```

Before the first entry into the **repeat** loop, N clusters exist (each data object forms its own separate cluster). At each loop pass, the number of clusters is reduced by one. Therefore, the algorithm ends after $N - 1$ passes when all data objects belong to the same cluster.

Agglomerative hierarchical clustering techniques use different criteria to decide locally at each step which two clusters to merge. This approach avoids the difficulty of solving the combinatorial optimization problem of clustering.

$d(\cdot,\cdot))_{Eukl}$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$
$\mathbf{x}_1$	0.00	0.24	0.22	0.37	0.34	0.23
$\mathbf{x}_2$	0.24	0.00	0.15	0.20	0.14	0.25
$\mathbf{x}_3$	0.22	0.15	0.00	0.18	0.28	0.11
$\mathbf{x}_4$	0.37	0.20	0.18	0.00	0.29	0.22
$\mathbf{x}_5$	0.34	0.14	0.28	0.29	0.00	0.39
$\mathbf{x}_6$	0.23	0.25	0.11	0.22	0.39	0.00

Table 1: Distances between 6 data objects.

However, once a decision is made to merge two clusters, it cannot be reversed later (there are some techniques that attempt to overcome this limitation).

The key operation of Algorithm 1 is to compute the *distance* $d(C_i, C_j)$ between two clusters C_i and C_j and it is the definition of this cluster distance that distinguishes the various agglomerative hierarchical techniques. Two common approaches are[1]:

- **single-linkage distance**

  $d(C_i, C_j) = \min\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$

  The distance between two clusters is equal to the distance between the two most similar data objects from C_i and C_j, respectively.

- **complete-linkage distance**

  $d(C_i, C_j) = \max\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$

  Here we use the distance between the two most distant data objects from different clusters.

Example 1 *Consider the Euclidean distances between six data objects in Table 1. Given the clusters* $C_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ *and* $C_2 = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$.

- *single-linkage:* $d(C_1, C_2) = 0.11$
- *complete-linkage:* $d(C_1, C_2) = 0.37$

Note that in agglomerative hierarchical clustering, the goal is independent of the distance used – in each step, the two clusters that have the smallest distance are merged. The difference is how that distance is defined.

Agglomerative hierarchical clustering is often represented graphically by a tree-like diagram called a *dendrogram*. A dendrogram shows both the cluster-subcluster relationships and the order in which the clusters were merged. On the one hand, a dendrogram gives a visual impression of the structure of the data. On the other hand, the tree can also be used to decompose the set X into a certain number of clusters. To do this, all that needs to be done in the dendrogram is to make a horizontal cut between two planes.

[1]There exist other – not considered in detail here – ways to define the distance between two clusters.

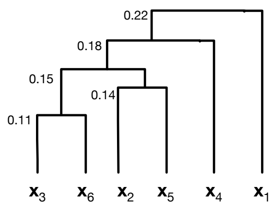


Figure 1: Dendrogramm based on single-linkage.

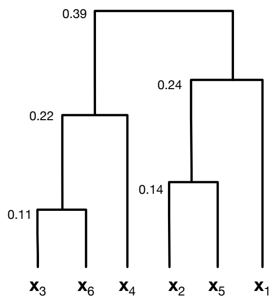


Figure 2: Dendrogramm based on complete-linkage.

Example 2 *To illustrate the behavior of hierarchical clustering with single or complete linkage, we will use the example data or distances between each data object from Table 1. Fig. 1 and Fig. 2 show the dendrograms of clusterings with single- and complete-linkage, respectively. The height at which two clusters are merged reflects the distance between the two clusters.*